Chapter Overview

• Behavior and Sequences
• Comparing Sequences
• Finding frequent subsequences
• Markov chains
• Hidden Markov-Chains
• Time series and feature-transformations
• Comparing time series
• Poisson-Processes
examples for player behavior

- sequence of moves in chess
- sequence of movement, action and interaction in a MMORPG
- sequence of orders to units in RTS Games

- conceptionally behavior consists of a sequence of possible actions
- Simplest models for behavior are strings or sequences.

**Definition:** Let $A=\{A_1, \ldots, A_n\}$ be a finite alphabet of $n$ possible player actions, then the $l$-Tuple $(a_1, \ldots, a_l) \in A \times \cdots \times A$ is a sequence of $l$ length over $A$.

**Remark:**
- Model describes only observations and does not differentiate between possible and impossible sequences.
- Model neglects the time between actions.
Example: SC II Zerg Rushes

While the build is a choose build, in that it needs to do damage an else you will be behind, it is made to transition into a standard game after the first 6 or 8 Zerglings. However it can be turned into all in simply by continuing to build Zerglings.

**Basic Build Order**

All the variations of this opening are shown below:

- **6 Pool**
  - 6 Spawning Pool
  - 5 Drone
  - 6 Drone
  - ★ 100% Spawning Pool, 3 pairs of Zerglings
  - 10 Zergling (Extractor Trick)
  - 11 Overlord

- **7 Pool**
  - 7 Spawning Pool
  - 6 Drone
  - 7 Drone
  - 8 Overlord
  - ★ 100% Spawning Pool, 3 pairs of Zerglings
  - a 4th pair of Zerglings when the Brain becomes available

- **8 Pool**
  - 8 Spawning Pool
  - 7 Drone
  - 8 Drone
  - 9 Overlord
  - ★ 100% Spawning Pool, 3 pairs of Zerglings
  - a 4th pair of Zerglings when the Brain becomes available

- **9 Pool**
  - 9 Spawning Pool
  - 8 Drone
  - 9 Drone
  - 10 Drone (Extractor Trick)
  - 11 Overlord
  - ★ 100% Spawning Pool, 3 pairs of Zerglings
  - a 4th pair of Zerglings when the Brain becomes available

- **10 Pool**
  - 10 Spawning Pool
  - 9 Drone
  - 10 Drone
  - 10 Overlord
  - 10 Drone (Extractor Trick)
  - ★ 100% Spawning Pool, 3 pairs of Zerglings
  - a 4th pair of Zerglings when the Brain becomes available

**Notes**

The only key difference in the variations, besides the number of drones and timing of Zerglings, is that a 6 Pool doesn’t allow for an Overlord to be made before the Zerglings are ready.

10 Pool allows you to start your Queen almost immediately after your initial 6 Zerglings are made.

**Spawning Pool timings**

- ★ 100% Spawning Pool
- 3 pairs of Zerglings
- a 4th pair of Zerglings when the Brain becomes available
Subsequences and Partitioning

- Which player is observed at a given time and for how long?
- The longer a player is observed, the less likely it becomes that another player behaves similarly.
- To find typical behavioral patterns a sequence is usually divided into subsequences.
- Windowing (partitions a sequence)
  Slide a window of length $k$ over the sequence and consider all subsequences. (here $k = 3$)
Subsequences and Partitioning

**problem:** A sequence of length $l$ has $l - (k-1)k$-elemental subsequences and many of those are irrelevant.

**idea:** Only sequences appearing with a certain frequency are of interest.

**Frequent Subsequence Mining**
Find all subsequences in a sequence database appearing more frequently than $\text{minsup}$. (cf. Frequent Itemset Mining)

⇒ length of the sequence is arbitrary.
⇒ search space is larger than the search space of itemset mining. (several occurrences of elements and orders)
Frequent Subsequence Mining

- frequency $fr(S,G)$ of $S$ in sequence $G$: count occurrence of $S$ in $G$
- relative frequency of $S$:
  \[
  \phi(S,G) = \frac{fr(S,G)}{|G| - |S| - 1}
  \]
- sequence description of $G$:
  \[
  \delta(G) = \{(S, \phi(S,G)) | S \in G\}
  \]
- mining sequential patterns is well explored
  $\Rightarrow$ many approaches and algorithms
Properties of a Suffix Tree $ST$ for the alphabet $A$ with sequence $G$ where $|G| = n$:

- to rule out ambivalence, words are padded with a terminal symbol ($A$), commonly $\$$.
- $ST$ has exactly $n+1$ leaf nodes numbered from 0 to $n$, on the way from the root to the leaf $i$ the suffix of length $n-i$ is filed.
- Edges represent elements of $A\{$$\} (uncompressed form), non-empty partial-sequences of $A\{$$\} respectively
- Edges, emanating from the same starting node, must begin with different elements of $A$.

Creation in $O(|\text{input string}|)$, Search in $O(|\text{query string}|)$
Suffix Trees

• example: alphabet \( A = \{\text{eat, hunt, seek, flee, defend}\} \)

• insert:
  \( S_1 = (\text{seek, hunt, eat, seek}) \)
  \( S_2 = (\text{seek, flee, hunt}) \)
Suffix Trees

- example: Alphabet $A = \{\text{eat, hunt, seek, flee, defend}\}$
- insert:

$S_1 = (\text{hunt, seek, eat, seek})$ $(\text{hunt, seek, eat, seek, }\$)$
$S_2 = (\text{seek, flee, hunt})$ $(\text{seek, flee, hunt, }\$)$

uncompressed variant:
- every edge is labeled with an element of $A\{\$\}$

compressed variant:
- combine sub-paths without branches into one edge
Suffix Trees

- example: Alphabet $A = \{\text{eat, hunt, seek, flee, defend}\}$
- insert:
  
  $S_1 = (\text{hunt, seek, eat, seek}) \ (\text{hunt, seek, eat, seek, \$})$
  
  $S_2 = (\text{seek, flee, hunt}) \ (\text{seek, flee, hunt, \$})$

uncompressed variant:
Every edge is labeled with an element of $A\{\$\}$

compressed variant:
combine sub-paths without branches into one edge
Suffix Trees

- example: Alphabet $A = \{\text{eat, hunt, seek, flee, defend}\}$
- insert:
  
  $S_1 = (\text{hunt, seek, eat, seek}) \ (\text{hunt, seek, eat, seek, } \$) \newline
  S_2 = (\text{seek, flee, hunt}) \ (\text{seek, flee, hunt, } \$)$
Suffix Trees

- example: Alphabet $\text{A} = \{\text{eat, hunt, seek, flee, defend}\}$
- sample queries:
  - Is $q$ a Suffix?
  - Is $q$ a Substring?
  - How often occurs $q$?
**Suffix Trees**

- Example: Alphabet $A = \{\text{eat, hunt, seek, flee, defend}\}$
- Sample request:
  - Is $q$ a Suffix?
    $\Rightarrow$ follow path $(q\$)$ starting at root,
    If reaching a leaf, then it is a Suffix
  - Is $q$ a Substring?
    $\Rightarrow$ follow path $(q)$ starting at root,
    If processing possible, then Substring
  - How often occurs $q$?
    $\Rightarrow$ follow path $(q)$ starting at root
    #leaves below terminal nodes
    $= \#\text{Occurrences}$
Interestingness of Subsequences

- interesting ≠ frequent

**common sequence**: select drones, collect crystals, train drone, ...

**but**: the first actions in SC II are almost always identical.

- number of frequent subsequences can be very large.
- most of which describe standard game plays.

- interestingness should be evaluated in relation to another attribute:
  - Map (Relating to a place)
  - Player (Relating to an individual)
  - Strategy (Relating to situation)
  - Combination of multiple relations (Map and Strategy ...)
Measures for Interestingness

use correlation measures:

- find a target variable: e.g. player_id
- find interesting events: e.g. boss-fights, flag bearer, ...
- find places triggering similar behavior: spawning points, flag delivery locations, boss encounter site, ...

example calculations:

- **Mutual Information**

\[
MI(S, \text{Player\_ID}) = \sum_{P \in \text{Players}} \sum_{S \in \{S_1, S_1\}} \Pr[S, P] \cdot \log \frac{Pr[S, P]}{Pr[S] \cdot Pr[P]}
\]

- **Odds Ratio**

\[
\text{oddsR}_s(G_1, G_2) = \frac{\phi(S, G_1)}{\phi(S, G_2)}
\]
Use of frequent subsequences

- **player identification**: use the occurrence of the k-“most interesting” partial sequences as vector space dimensions.  
  (here interesting = highest MI with player_id)  
  => describe players as vectors of observed subsequences.

- **search locations specific behavior**: compare the incidence of actions on the map to the amount of actions in a given location. (Odds-Ratio)
Comparing two Sequences

given: Alphabet A and a sequence database
\[ DB = \{(x_1, \ldots, x_k) | k \in \mathbb{IN} \land x_i \in A \text{ for } 1 \leq i \leq k\} \]

task: compute the similarity of \( S_1, S_2 \in DB \).

Hamming Distance: number of different entries over all positions.

For 2 sequences with \(|S_1|=|S_2|=k\):

\[
Dist_{Ham}(S_1, S_2) = \sum_{i=0}^{k} \begin{cases} 
0 & \text{if } s_{1,i} = s_{2,i} \\
1 & \text{else} 
\end{cases}
\]

Remark: For sequences of different length, the shorter sequence is filled with the gap symbol „-“.

eexample: \( S_1 = (A,B,B,A,B) \) und \( S_2 = (A,A,A,A,A) \)

\[
\begin{array}{c}
(A,B,B,A,B) \\
(A,A,A,A,A)
\end{array}
\quad Dist_{Ham}(S_1, S_2)=3
\]
Levenshtein Distance

- **Hamming Distance**: Computing the minimum cost to transform $S_1$ into $S_2$. Only substitutions of single elements are allowed in doing so. (Turn B into A.)

- **Hamming Similarity**: Counts the number of similar elements.

- **idea**: Extend the allowed transformations to include deletion and insertion of symbols.

- **Levenshtein Distance**: Minimum expense to transform $S_1$ into $S_2$ using 3 operations *Delete, Insert* and *Substitute*.

$$\left\{ \begin{array}{l}
(A,B,B,A,B) \\
(A,A,B)
\end{array} \right\} \left\{ \begin{array}{l}
(A,B,B,A,B) \\
(A,-,-,A,B)
\end{array} \right\} Sim_{Lev}(S_1,S_2)=3$$
Calculating Levenshtein Distance

given: Two sequences S1, S2 over the alphabet A with \(|S1|=n\) and \(|S2|=m\).

**task:** \(Dist_{Lev}(S1,S2)\)

Calculating Levenshtein Distance with dynamic programming:
Let \(D\) be a \(n \times m\)-Matrix over \(\mathbb{IN}\) with:

\[
D_{0,0} = 0
\]

\[
D_{0,i} = i, \quad 0 \leq i \leq n
\]

\[
D_{j,0} = j, \quad 0 \leq j \leq m
\]

\[
D_{i,j} = \min\begin{cases} 
D_{i-1,j-1} + 0, & \text{falls } s_{1i} = s_{2j} \\
D_{i-1,j-1} + 1, & (\text{Substitution}) \\
D_{i,j-1} + 1, & (\text{Insertion}) \\
D_{i-1,j} + 1, & (\text{Deletion}) 
\end{cases}
\quad \text{für } 1 \leq i \leq n, \quad 1 \leq j \leq m
\]

After construction of matrix \(D\), \(D_{n,m}\) contains the Levenshtein-distance between both input sequences.
Example Levenshtein Distance

S1 = auto, S2 = ute

\[ \text{Dist}_{\text{Lev}} (S1, S2) = 2 \]
Edit Distances

• generalization of Levenshtein-Distances:
  • different cost matrix: substitution costs 4, deletion 1, insertion 2..
  • more operations:
    • transposing order
      $$(A,B,B,A,B) \rightarrow (A,B,A,B,B)$$
    • duplicating, …
      $$(A,B,B,B,B) \rightarrow (A,B,)$$

$$1 \text{ transposition}$$
$$3 \text{ duplicates of } B$$

• costs may differ for different values:
  $$\text{Subst.}(A,B) \neq \text{Subst.}(A,Z)$$

• works for sequences based on real-valued alphabets, for example: For $$A = IR: \text{Subst}(5,1) = |5-1|$$
Markov Chains and Sequences

- sequences of actions are subject to certain rules
- modeling with finite automatons (testing sequence for validity)
- Markov chains are probabilistic automatas:
  - allowed state transitions
  - probability distributions for state transitions.
- 1st order Markov assumption: The state at time \( t+1 \) depends solely on the state at time \( t \).
- the order of a Markov chain is the number of predecessor states on which the choice of the next state might depend.
**First Order Markov-Chains**

**definition:** A Markov chain $M$ is defined for a state set $A$ and a stochastic transition-matrix $|A| \times |A| = D$.

**explanations:**
- $A$ may contain a start- and a absorption-state (Modeling Start and End)
- stochastic Matrix: rows add up to 1.
  (row $i$ contains the distribution of successors for state $i$)

**example:**

$$p(ACBB) = P(A | -) \cdot P(C | A) \cdot P(B | C) \cdot P(B | B) \cdot P(- | B)$$

$$= 0.3 \cdot 0.15 \cdot 0.4 \cdot 0.7 \cdot 0.4 \cdot 0.1$$
Hidden Markov Models

**training a Markov chain:**
- break the training sequence down into 2-grams and determine the relative frequency.
  (How often is A followed by B?)

\[
P(B \mid A) = \frac{fr(AB)}{fr(A)}
\]

**problem:**
- observations often do not match the observed behavior:
  - action log is available, but game-play has to be analyzed
  - incorrect execution obfuscates actual intentions
  - analysis of an AI state changes
    (observed actions may be employed in different states)
**Definition:** A Hidden Markov Model $M$ is defined by a state set $A$, a stochastic transition matrix $|A| \times |A| = D$, an observation set $B$ and a stochastic output-matrix $|A| \times |B| = F$.

**Example:** $A = \{A, B, C\}$, $B = \{1, 2, 3\}$

$P(122)$: define all possible state triples, generated by $122$:

BAA, BAC

\[
P(122) = P(BAA) \cdot P(122 | BAA) + P(BAC) \cdot P(122 | BAC)
\]
Use of HMM

- **Evaluation**: How likely is an observation $O=(o_1, \ldots, o_k)$ with $o_i \in B$ for the HMM $(A, B, D, F)$?
  
  *(Forward Estimation)*

- **Recognition**: Given the observation $O=(o_1, \ldots, o_k)$ and the HMM $(A, B, D, F)$ which sequence $(s_1, \ldots, s_k)$ with $s_i \in A$ gives the best explanation for $O$? *(Viterbi-Algorithm)*

- **Training**: Given the observation $O=(o_1, \ldots, o_k)$, how can we modify $D$ and $F$ to maximize $P(O|A, B, D, F)$?
  
  *(Baum-Welch Estimation)*
Evaluation: Forward Variables

given: \( O=(o_1, \ldots, o_k) \) and \( (A,B,D,F) \)

task: \( P(O|(A,B,D,F)) \)

naive solution: calculate \( P(O|S) \) for all \( k \)-elemental sequences \( S \) on \( A \).
(number grows exponentially with \( k \))

improved solution: utilize Markov assumption

define forward-variable \( \alpha_j(t) \) as

\[
\alpha_j(t) = P(o_1, o_2, \ldots, o_t, s_t = a_j \mid (ABDF))
\]

calculation by induction:

\[
\alpha_j(1) = d_{-,j} \cdot f_{j,o_1}, \quad 1 \leq j \leq |A|
\]

\[
\alpha_j(t + 1) = \left( \sum_{i=1}^{|A|} \alpha_i(t) \cdot d_{i,j} \right) \cdot f_{j,ot+1}, \quad 1 \leq t \leq k - 1
\]

calculating with \( |A|^2 \cdot k \) operations:

\[
P(O \mid (A, B, D, F)) = \sum_{i=1}^{|A|} P(O, s_t = a_i \mid (A, B, D, F)) = \sum_{i=1}^{|A|} \alpha_i(k)
\]
Recognition: Viterbi Algorithm

given: \( O = (o_1, \ldots, o_k) \), and Model \((A, B, D, F)\).

task: \( S = (s_1, \ldots, s_k) \), which maximizes \( P(O|S, (A, B, D, F)) \).

• define \( \delta(t) \) as the highest probability of a sequence on \( A \) of length \( t \) for the observation \( O \).

\[
\delta_j(t) = \max_{s_1, \ldots, s_{t-1}} P(s_1, \ldots, s_{t-1}, O \mid (A, B, D, F))
\]

• calculation by induction

\[
\delta_j(1) = d_{-,j} \cdot f_{j, o_1}, \quad 1 \leq j \leq |A| \\
\delta_j(t + 1) = \left( \max_{1 \leq i \leq |A|} \left( \delta_i(t) d_{i,j} \right) \right) \cdot f_{j, o_{t+1}}, \quad 1 \leq j \leq k - 1
\]

\[
\psi_j(1) = 0, \quad 1 \leq j \leq |A| \\
\psi_j(t + 1) = \text{arg max}_{1 \leq i \leq |A|} \left( \delta_i(t) d_{i,j} \right), \quad 1 \leq j \leq k - 1
\]

• similar to forward algorithm, but more efficient since only the best solution is pursued.
Backward Variables

analogously to Forward-Variable a Backward-Variable can be defined, used in training the HMM.

define Backward-Variable $\beta_j(t)$ as

$$\beta_j(t) = P(o_{t+1}, ..., o_k \mid s_t = a_j, (ABDF))$$

Calculation by Induction:

$$\beta_i(k) = 1 \quad , 1 \leq i \leq |A|$$

$$\beta_i(t - 1) = \sum_{j=1}^{\lfloor A \rfloor} d_{i,j} \cdot f_{j,o_t} \cdot \beta_j(t) \quad , 2 \leq t \leq k$$
Training: Baum-Welch Estimation

given: \( O=(o_1, \ldots, o_k) \), \( A \) and \( B \).

task: \( D, F \), maximizing \( P(O|(A,B,D,F)) \).

- Locally optimize solution with \textit{Expectation Maximization (EM)}

Define \( \xi_{i,j}(t) \) as the likelihood of being in state \( a_i \) at the point in time \( t \) and being in state \( a_j \) at the point in time \( t+1 \):

\[
\xi_{i,j}(t) = P(s_t = a_i, s_{t+1} = a_j \mid O, (A, B, D, F)) = \frac{\alpha_i(t) \cdot d_{i,j} \cdot f_{j,o_{t+1}} \beta_j(t+1)}{P(O \mid (A, B, D, F))} = \frac{\alpha_i(t) \cdot d_{i,j} \cdot f_{j,o_{t+1}} \beta_j(t+1)}{\sum_{k=1}^{\mid A \mid} \sum_{l=1}^{\mid A \mid} \alpha_k(t) \cdot d_{k,l} \cdot f_{l,o_{t+1}} \beta_j(t+1)}
\]

- Define \( \gamma_i(t) \) as the probability of being in state \( a_i \) at the point in time \( t \):

\[
\gamma_i(t) = \sum_{j=0}^{\mid A \mid} \xi_{i,j}(t)
\]
Training: Baum-Welch Estimation

- \( \sum_{t=1}^{k-1} \xi_{i,j}(t) \) equals the expected number of state transitions from \( a_i \) to \( a_j \).
- \( \sum_{t=1}^{k-1} \gamma_i(t) \) equals the expected number of state transitions from \( a_i \) to other states.
- Parameters are being recomputed as follows:
  
  \[
  d_{-,a_i} = \gamma_i(1), \quad d_{i,j} = \frac{\sum_{t=1}^{k-1} \xi_{i,j}(t)}{\sum_{t=1}^{k-1} \gamma_i(t)}, \quad f_{j,b_i} = \frac{\sum_{t \in \{ t \mid a_t = b_i \}} \gamma_i(t)}{\sum_{t=1}^{k-1} \gamma_i(t)}
  \]

- Training happens in alternating steps
  - Calculate of \( \gamma_i(t), \xi_{i,j}(t) \) and \( P(O|I(A,B,D,F)) \)
  - Updates of \( D \) and \( F \) (updates see above)

- Algorithm terminates when \( P(O|I(A,B,D,F)) \) grows less than .
Real-Value Sequences

- **so far**: Alphabet is a discrete domain
- Sequences can also be created based on real-value domains, for example $IR^d$.
- Frequent Pattern Mining on real-value domains is usually impossible.
- Comparing 2 real-value sequences on domain D with a distance function $dist: D \times D \to IR^+$.
  - Analogous to Hamming Distance one can determine the sum of distances for every position of the sequence.
  $$dist_{sequ}(S_1, S_2) = \sum_{i=1}^{||S_1||} dist(s_{1,i}, s_{2,i}) + (||S_2|| - ||S_1||) \cdot \varphi, \quad \text{für } |S_2| \geq |S_1|, \varphi \in IR^+$$
  - Extension of edit distance is also possible: Substitution cost for $v$ and $u$ correlates to $dist(v,u)$.
  - (More details follow later for Dynamic Time Warping)
Time series

- **so far**: sequences model the order of actions, but not the points in time.

  **but**: in real time games timing is essential.

  ⇒ RTS games: build order are only effective if they can be realized in minimal time.

  ⇒ in MMORPGs the damage caused depends on the number of actions per time unit.

  ⇒ chess with chess clock: a move is also measured by the time needed to think.

- **time series**: Let $T$ be a domain to model time and let $F$ be an object presentation, then:

  $Z=\{(x_1,t_1),\ldots,(x_l,t_l)\} \in (F \times T) \times \ldots \times (F \times T)$ is a time series of length $l$ on $F$. 
Examples for Time Series

- **SC2-Logs**: time series on discrete actions

<table>
<thead>
<tr>
<th>Time</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00</td>
<td>TSLHyuN  Select Hatchery (10230)</td>
</tr>
<tr>
<td>0:00</td>
<td>TSLHyuN  Select Larva x3 (1027c,10280,10284), Deselect all</td>
</tr>
<tr>
<td>0:00</td>
<td>TSLHyuN  Train Drone</td>
</tr>
<tr>
<td>0:01</td>
<td>TSLHyuN  Train Drone</td>
</tr>
<tr>
<td>0:01</td>
<td>TSLHyuN  Select Drone x6 (10234,10238,1023c,10240,10244,10248), Deselect all</td>
</tr>
<tr>
<td>0:01</td>
<td>TSLHyuN  Right click; target: Mineral Field (10114)</td>
</tr>
<tr>
<td>0:01</td>
<td>TSLHyuN  Deselect 6 units</td>
</tr>
<tr>
<td>0:02</td>
<td>TSLHyuN  Right click; target: Mineral Field (10170)</td>
</tr>
</tbody>
</table>

- **Network-Traffic**:
  - used in bot detection
  - estimating game intensity

![Network Traffic Diagram](image)
Preprocessing Time series (1)

offset translation

- similar time series with different offsets
- shifting all time series around the mean $MW$:
  
  $\forall i \mid o_i = o_i - MW(o)$

\[
\begin{align*}
q &= q - MW(q) \\
o &= o - MW(o) \\
dist(q,o) &= ???
\end{align*}
\]
preprocessing time series (2)

scaling amplitudes

- time series with similar progression but different amplitudes
- shifting the time series around the mean ($MW$) and normalizing the amplitude by standard deviation (StD):

\[
o_i = \frac{o_i - MW(o)}{StD(o)}
\]

\[
q = \frac{q - MW(q)}{StD(q)}
\]

\[
o = \frac{o - MW(o)}{StD(o)}
\]
preprocessing time series (3)

**linear trends**

- similar time series with different trends
- Intuition:
  - determine regression line
  - move time series by means of this line

- offset translation + amplitudes scaling

- offset translation + Amplitudes scaling 
  + linear trend-removal
rectifying noise

- similar time series with a large amount of noise
- smoothing: determine for every value $o_i$ the mean over all values $[o_{i-k}, \ldots, o_i, \ldots, o_{i+k}]$ for a given $k$. 
Discrete Fourier Transformation (DFT)

**idea:**
- describe arbitrary periodic functions as weighted sum of periodic *base functions* with different frequencies. A time series turns into a vector of constant length.
- base functions: sin and cos

![Diagram](image_url)
**Fourier’s theorem:** A periodic function (which is reasonable continuous) may be expressed as the sum of a series of sine and cosine terms with a specific amplitude.

**Properties:**
- Transformation does not change a function, only the presentation
- Transformation is reversible $\Rightarrow$ inverse DFT
- Analogy: change of base in vector calculation
Discrete Fourier Transformation (DFT)

formal:
• given a time series of length $n$: $x = [x_t]$, $t = 0, \ldots, n - 1$
• the DFT of $x$ is a sequence $X = [X_f]$ of $n$ complex numbers for the frequencies $f = 0, \ldots, n - 1$ with

$$X_f = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cdot e^{-\frac{i \cdot 2 \pi \cdot f \cdot t}{n}} =$$

$$\frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cos\left(\frac{2 \cdot \pi \cdot f \cdot t}{n}\right) - i \cdot \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \sin\left(\frac{2 \cdot \pi \cdot f \cdot t}{n}\right)$$

where $i$ identifies the imaginary unit viz. $i^2 = -1$.
• the real part indicates the share of the cosine functions, whereas the imaginary part indicates the share of sine functions of the frequency $f$. 
Discrete Fourier Transformation (DFT)

• the inverse DFT restores the original signal:

\[ x_t = \frac{1}{\sqrt{n}} \sum_{f=0}^{n-1} X_f \cdot e^{\frac{i \cdot 2 \cdot \pi \cdot f \cdot t}{n}} \]

\( t = 0, \ldots, n - 1 \) (t: points in time)

\([x_t] \leftrightarrow [X_f]\) describes a Fourier-Paar,

viz. DFT([x_t]) = [X_f] and DFT\(^{-1}([X_f]) = [x_t]\).

• the DFT is a linear map, viz. from \([x_t] \leftrightarrow [X_f]\)

and \([y_t] \leftrightarrow [Y_f]\) follows:

• \([x_t + y_t] \leftrightarrow [X_f + Y_f]\) and

• \([ax_t] \leftrightarrow [aX_f]\) for a Scalar \(a \in \mathbb{R}\)

• energy of a sequence

• energy \(E(c)\) of \(c\) is the square of the amplitude: \(E(c) = |c|^2\).

• energy \(E(x)\) of a sequence \(x\) is the sum of all energies

of the sequence:

\[ E(x) = \|x\|^2 = \sum_{t=0}^{n-1} |x_t|^2 \]
Discrete Fourier Transformation (DFT)

**Parseval’s theorem:** Energy of a signal in a time range equals the energy in the frequency range.

**Formal:** Let $X$ the DFT of $x$, then follows:

$$\sum_{t=0}^{n-1} |x_t|^2 = \sum_{t=0}^{n-1} |X_f|^2$$

- consequence from Parseval’s theorem and the DFT’s linearity: The euclidean distance of two signals $x$ and $y$ correspond in time and frequency range:

$$\| x - y \|^2 = \| X - Y \|^2$$

„Time range (-space)“    „Frequency range (-space)“
Discrete Fourier Transformation (DFT)

**Basic Idea of query processing:**

The euclidean distance is used as a sequence’s similarity function:

\[ \text{dist}(x, y) = \|x - y\| = \sqrt{\sum_{i=0}^{n-1} |x_i - y_i|^2} \]

- Parseval’s theorem allows for distances to be calculated in the frequency range instead of the time range: \( \text{dist}(x,y) = \text{dist}(X,Y) \)
- In practice, the lowest frequencies are the most important.
- The first frequency coefficients contain the most important information.
- For indexing, the transformed sequences are shortened, for \([X_f], f = 0, 1, \ldots, n - 1\) coefficients only the first \(c\) coefficients \([X_f < c], c < n\) are indexed.

\[ \text{dist}_{c}(x, y) = \sqrt{\sum_{f=0}^{c-1} |x_f - y_f|^2} \leq \sqrt{\sum_{f=0}^{n-1} |x_f - y_f|^2} = \text{dist}(x, y) \]

- For the index, a lower bound of the true distance can be calculated:
  - Filter-refinement:
    - Filter step is based on shortened time series (index assisted)
    - Refinement step determines true hits on complete time series
Distances of Time Series

**problems**: Which points in time are to be compared?

- offset at the beginning: S2 is shifted in time to S1.

- clocking of reading: points in time of measuring differ.

- length of time series: measuring periods differ.

- time series with the same clocking and length can be compared as vectors. (dimension = point in time)

\[ \text{Dist}_{\text{timeseries}}(S1, S2) = \sum_{t=1}^{T} \text{dist}_{\text{obj}}(s_{1t}, s_{2t}) \]

- for variable length, clocking and offsets: adaption of edit-distance for sequences \(\Rightarrow\) **Dynamic Time Warping**
Dynamic Time Warping Distanz

calculation:
• given: time series $q$ and $o$ of different length
• find mapping of all $q_i$ to $o$ with minimal expense

Search matrix
Dynamic Time Warping Distance

Search Matrix

- All possible mappings \( q \) to \( o \) can be interpreted as a „warping“ path within the search matrix
- Of all these mappings, we search for the path with the lowest cost

\[
DTW(q,o) = \min \left\{ \sqrt{\sum_{k=1}^{K} w_k} / K \right\}
\]

- Dynamic Programming
  ⇒ Run-time \( (n \cdot m) \)
  (see Edit Distances)
Approximate Dynamic Time Warping Distance

idea:

- approximate the time series
  (compressed representation, Sampling, ...)
- calculate DTW for the approximates
Problem: How is the time of the next action modeled?
⇒ statistic models for the time between two events is necessary.

⇒ time is a continuous variable ⇒ probability density function

⇒ task: compute the probability for the next event \( e \) occurring within the time frame \( t + t \).

⇒ the cumulative probability density function describes this probability
Homogeneous Poisson Processes

- simplest process to model time
- points in time between 2 events are exponentially distributed
- probability density of the exponential distribution: \( p_\lambda(x) = \lambda \cdot e^{-\lambda \cdot x} \)
- integration yields the cumulative density function describing the probability of the next action happening in the time interval between 0 … x.

\[
P_\lambda(x) = \int_0^x p_\lambda(t)\,dt = 1 - e^{-\lambda \cdot x}
\]

Density function of the exponential distribution

Accumulated density function of the exponential distribution
Parameter assessment

given: A training set of points in time \( X = \{x_1, \ldots, x_n\} \), which are distributed exponentially.

task: The most likely value for the intensity parameter.

Approximation with Maximum Likelihood

\( \Rightarrow \) Search the value of \( \lambda \) with the highest probability of generating \( X \).

Likelihood function \( L \) for Sample \( X \):

\[
L_X(\lambda) = \prod_{i=1}^{n} \lambda \cdot e^{-\lambda \cdot x_i} = \lambda^n \cdot e^{-\lambda \cdot \sum_{i=1}^{n} x_i} = \lambda^n \cdot e^{-\lambda \cdot n \cdot E(X)}
\]

mit \( E(X) = \frac{\sum_{i=1}^{n} x_i}{n} \)

Differentiate the log-likelihood for \( \lambda \) and set the gradient to zero:

\[
\frac{d}{d\lambda} \ln L(\lambda) = \frac{d}{d\lambda} (n \cdot \ln(\lambda) - \lambda \cdot n \cdot E(X)) = \frac{n}{\lambda} - n \cdot E(X)
\]

\( \Rightarrow \lambda^* = \frac{1}{E(X)} \)
Learning Goals

• Sequences and time series
• Frequent Subsequence Mining with Suffix-Trees
• Distance measuring sequences
  • Hamming Distance
  • Levenshtein Distance
• Markov-Chains
• Hidden Markov chains
• Time series and preprocessing steps
• Dynamic Time Warping
• Poisson processes
Literature

